TinyML Project - Road Sign Detection: Report

Narasimha Atreya, Praneeth Aluru, Sardar Karan, Anvesh Avirneni

***Abstract*—This report is a description of the project using a portenta micro controller. We have developed a model which could classify different road signs.**

***Index Terms*—portenta, micro-controller, model, machine-learning, artificial neural networks**

I. Introduction

A. Micro Controller

A microcontroller is a simple computer that is designed to run a single basic program on a continuous basis. Microcontrollers are single-chip microcomputers that are very small and self-contained. They are frequently intended to perform a single automated task in a single device, according to the user's pre-programmed instructions. They're programmed to do the same thing over and over. This is referred to as an embedded application, as opposed to the more versatile, general-purpose applications handled by entire microprocessors and CPUs.

A microprocessor is one of the key components of microcontrollers, however it's usually a much simpler and dynamic CPU than most standalone MPs. This is because most microcontrollers are only capable of performing a single, highly specialized task. This means it doesn't need the full set of capabilities that a typical microprocessor can offer. This microcontroller and PCB-based equipment combination can be used to control, monitor, and impact the behavior of a wide range of systems and components. The breadth of their distinct busses, which is the width of their data pipelines, is the primary difference between the three microcontroller kinds.

This specification is very important that a determines the speed and mathematical precision of a microcontroller. To perform 16-bit or 32-bit calculations, an 8-bit microcontroller will require more bus accesses and instructions. So, that's why it will reach at the answer significantly more slowly than a 16-bit or 32-bit MCU. We get the similar constraint when we have with a slower CPU rather than a faster, more powerful one in computing terms. These considerations will have an impact on the programming languages that can be used with a microcontroller unit, as well as the languages that can be used with it. Microcontrollers are generally compatible with a variety of programming languages, whether it's C++, Python, R, or Arduino, though the specifications will vary depending on the device. 8-bit microcontrollers have long been seen to be the most basic and cost-effective solution, however their capabilities are limited in several situations. Although 16-bit and 32-bit microcontrollers are more expensive than 8-bit and 16-bit microcontrollers, they offer equivalent performance. [1]

Somya D. Mohanty is with the Department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC, 27412 USA e-mail:sdmohant@uncg.edu.

Narasimha Atreya is a student with the department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC, 27412 USA e-mail:n_avadhanul@uncg.edu.

Praneeth Aluru is a student with the department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC, 27412 USA e-mail:p_aluru@uncg.edu.

Anvesh Avirneni is a student with the department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC, 27412 USA e-mail:a_avirneni@uncg.edu.

Sardar Karan Singh is a student with the department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC, 27412 USA e-mail:k_sardar@uncg.edu.

B. Working of Micro Controllers: -

When used as part of a functioning circuit in a device or system, a microcontroller board can sense, monitor, and respond to various events, behaviors, or input signals that it detects from linked components and its environment.

In response to certain input criteria, a microcontroller might be configured to push a specific type of output signal or behavioral control. This could entail completing duties such as:

- In response to a touch-based user demand, an LED or OLED display is illuminated.
- Temperature-sensing applications that play lights and sounds, as well as other types of alerts and warning systems
- Responding to the requirement for a motor in a pump or other mechanical device to turn on or off.
- Adjusting the gyroscope for tilt, balance, and velocity.

C. Arduino BLE 33:-

The Arduino Nano 33 BLE (without headers) is a 3.3V compliant board with the lowest possible compact factor: 45x18mm! The Arduino Nano 33 BLE is a brand-new board with a familiar compact factor. It has an inbuilt 9-axis inertial sensor, making it suitable for wearable devices as well as a wide range of scientific experiments requiring short-range wireless communication.

The Arduino Nano 33 BLE is an upgrade of the original Arduino Nano, but with a much more powerful processor from Nordic Semiconductors, the nRF52840, a 32-bit ARM® Cortex®-M4 CPU running at 64 MHz. This will let you to write larger programs than the Arduino Uno (it has 1MB of program memory, which is 32 times greater), and it will also allow you to use a larger screen. The Nano 33 BLE is equipped with a 9-axis inertial measurement unit (IMU), which contains a 3-axis resolution accelerometer, gyroscope, and magnetometer. As a result, the Nano 33 BLE is ideal

for sophisticated robotics experiments, fitness trackers, digital compasses, and more.
If you've worked with Arduino Nano before, the Nano 33 BLE is a pin-equivalent replacement. Your code will still work, but keep in mind that it runs on 3.3V. This means you'll need to make changes to your original design if it's not 3.3V compliant. A faster processor, a micro-USB port, and a 9-axis IMU are the key differences from the old Nano.
The Nano 33 BLE's communications chipset may function as a Bluetooth® Low Energy and Bluetooth® client and host device. In the realm of microcontroller platforms, this is rather unique. [2]

D. Portenta h7

The Portenta H7 has the same physical factor as the Arduino MKR, but adds the Portenta family's 80-pin high-density connection. It can be programmed using high-level languages and AI, and its programmable hardware can conduct low-latency tasks. The main features of portenta is that its can run on high level code along with real time masks. It is feasible, for example, to run Arduino built code alongside MicroPython code and have both cores communicate with each other. The Portenta has two modes of operation: it can serve as an embedded microcontroller board or as the main processor of an embedded computer. Use the Portenta Vision Shield, for example, to transform your H7 into an industrial camera that can run real-time machine learning algorithms on live video streams.

Highly Configurable Design:- Portenta H7 has been designed with high configuration. It provides lot of options for ordering different boards with different configurations of memory. [3]

Use Portenta when performance is critical, such as in the following scenarios: [4]

- Industrial machinery of the highest caliber
- Instruments for the laboratory
- Vision in a computer
- PLCs
- User interfaces that are suitable for usage in the industry
- Mission-critical devices are controlled by robotics.
- Computer with a fixed location
- Computation of high-speed booting (ms)

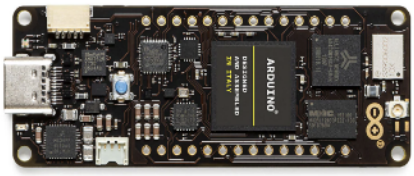

Fig. 1. Road Sign: Portenta H7
[18]

E. Portenta Vision Shield

The Portenta Vision Shield gives your Arduino Portenta industry-standard functionalities. This hardware add-on allows you to run embedded computer vision programs, connect to the Arduino Cloud or your own infrastructure through wireless or Ethernet, and activate your system when sound events are detected. [5]

- 320x320 pixels camera sensor: use one of the cores in Portenta to run image recognition algorithms using the OpenMV for Arduino editor
- 100 Mbps Ethernet connector: get your Portenta H7 connected to the wired Internet
- two on-board microphones for directional sound detection: capture and analyse sound in real time
- JTAG connector: perform low-level debugging of your Portenta board or special firmware updates using an external programmer
- SD-Card connector: store your captured data in the card, or read configuration files

F. How does Portenta help in our project ?

Arduino Portenta H7 is a low-power microcontroller which has Industry-rated features which will let us run the embedded computer vision applications. The specifications of vision shield is:

- It has a 320 x 320 pixels camera sensor which is used to run algorithms related for image processing.
- It has a 100 Mbps Ethernet connector to connect the board to the wired internet.
- It has two on-board microphones for audio detection.
- It also does have a SD-Card connector to store any captured data in the card or for reading any configuration files.

The main use of portenta here is to make it work as a micro-controller with our desired functionality based on the kind of program with which we build it. We can achieve that goal by using Machine Learning or Deep Learning. In machine learning /deep learning, we create models which would automate the prediction of any kind of new values given to it. To make it happen, we first pre-process the data to be more suitable for modelling to get high prediction accuracies. In our project, we created train and validation data which has several arguments that would do data augmentation i.e flipping the images horizontally/vertically, rotating etc., After that, we created a deep learning model which was showed above to and started training the model for which we got the varying accuracies and losses. After getting the ideal model for which we got the highest accuracy, we reduced the size of the model to make it run in the memory of Portenta Vision Shield. To achieve that, we did quantization and pruning of the model to generate a file type called "tflite" file which is a compressed model file which would fit in the memory of portenta and works like a micro-controller with the functionality by which it was trained. For our project,

it was trained to detect or classify the road signs from one another when the portenta camera was pointed towards the road sign.

After generating the tflite file, we will connect the portenta to the system and just open the "Open MV" IDE which is used for deployment purpose where the compressed model i.e the tflite model and a file of class labels with the proper order in which it was trained should be dropped in the portenta flash memory. When the code is uploaded into the portenta it should be able to perform kind of task which we want. The programming language accepted by Open MV is Python. The code which we have first takes inputs from the portenta camera and directly to the model. After getting the predictions, there is a calculation of confidence intervals for all the predictions. We consider only that prediction which has the highest confidence values and finally print that value saying that is the predicted value.

Now, the portenta works like a road sign detector which just captures the image of the road sign and directly classifies it as the appropriate road sign.

II. Data description

We have taken 8 road sign for training the model.
The number of images for each class is 500.
So a total of 4000 images are used to train and validate the model.
The image ratio is 320x240.

The list of road signs we used for the model:

- Divided Highway Ends
- Keep left
- Keep Right
- Lane Drop
- Merging traffic
- Slippery when wet
- Two way traffic
- Winding road

Fig. 2. Road Sign: Divided Highway ends

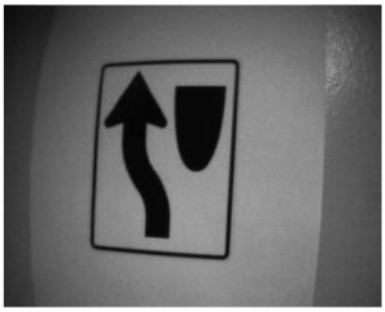

Fig. 3. Road Sign: Keep Left

Fig. 4. Road Sign: Keep Right

Fig. 5. Road Sign: Lane Drop

Fig. 6. Road Sign: Merging Traffic

Fig. 7. Road Sign: slippery when wet

Fig. 8. Road Sign: Slippery When Wet

Fig. 9. Road Sign: Winding Road

The images are taken using portenta H7.

III. Data manipulation

A. *ImageDataGenerator*

Image augmentation is a technique of applying different transformations to original images which results in multiple transformed copies of the same image. Each copy, however, is different from the other in certain aspects depending on the augmentation techniques you apply like shifting, rotating, flipping, etc.

Applying these small amounts of variations on the original image does not change its target class but only provides a new perspective of capturing the object in real life. And so, we use it is quite often for building deep learning models.

These image augmentation techniques not only expand the size of your dataset but also incorporate a level of variation in the dataset which allows your model to generalize better on unseen data. Also, the model becomes more robust when it is trained on new, slightly altered images. [6]

1) Augmentation techniques with Keras ImageDataGenerator class:

- Random Rotations
- Random Shifts
- Random Flips
- Random Brightness
- Random Zoom

IV. Model

A machine learning model is a computer program that has been trained to recognize specific patterns. You train a model on a set of data and give it an algorithm to use to reason about and learn from that data.

Once the model has been trained, you can use it to reason over data it hasn't seen before and make predictions about it. Consider the following scenario: you want to create an app that can understand a user's emotions based on their facial expressions. You can train a model by feeding it photos of faces each labelled with a different emotion, and then use that model in an application to recognize any user's emotion.

In the same way in our current project, we are using road sign symbols as our model where in our model would detect the type of road sign and predict the symbol. [7]

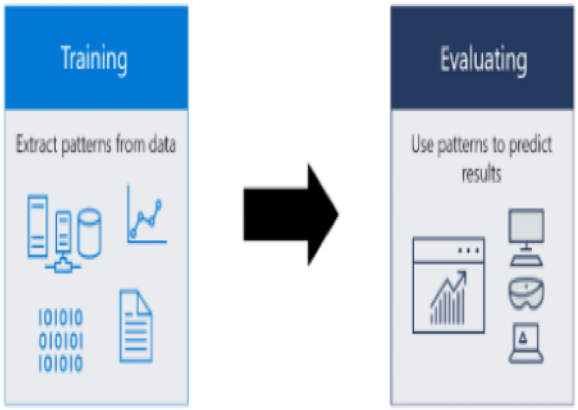

Fig. 10. Model

The Common Properties of good machine learning model:

- They require consistent results and entail a repeated judgment or evaluation that you want to automate.
- Explicitly describing the solution or criteria that led to a conclusion is difficult, if not impossible.
- You have labeled data or examples of how to characterize a scenario and map it to the correct outcome.

V. The Neural Networks Model

Structure of Neural Networks:

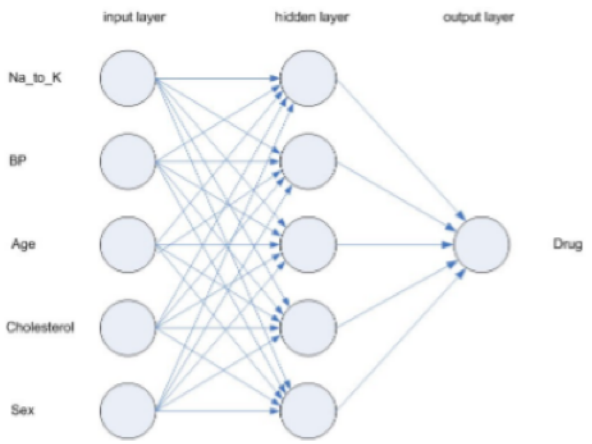

Fig. 11. Neural Networks Model

Neural networks are simplified representations of how the nervous system functions. Neurons are the basic units, which are often grouped into layers. As shown in above figure.

A neural network is a simplified representation of how the human brain processes data. It simulates a vast number of interconnected processing units that seem like abstract versions of neurons.

The processing units are stacked one on top of the other. An input layer with units representing the input fields, one or more hidden layers, and an output layer with a unit or units representing the target field are the three pieces of a neural network (s). The units are linked together using a variety of connection strengths (or weights). The first layer receives input data, and values are transferred from each neuron to the following layer's neurons. The output layer will eventually offer a result.

The network learns by studying individual data, providing a forecast for each record, and adjusting the weights if the prediction is erroneous. This process is repeated several times, and the network's predictions improve until one or more of the halting criteria is reached.

Initially, all weights are random, and the results obtained from the net are almost certainly absurd. Training is how the network learns. The network is repeatedly provided with examples for which the output is known, and the network's responses are compared to the known outcomes. The weights are gradually changed when the information from this comparison is transmitted back across the network. The network grows more accurate at duplicating known outcomes as training advances. The network can be used in future scenarios when the outcome is unclear once it has been trained. [8]

There are different types of neural network models some of them are: [9]

A. Multilayer Perceptrons

A feed-forward artificial neural network called a multilayer perceptron (MLP) is a type of feed-forward artificial neural network. A single neuron model that is a predecessor to a bigger neural network is referred to as a perceptron.The input layer, the hidden layer, and the output layer are the three main layers of nodes in an MLP. Every node in the hidden and output layers is treated as a neuron with a nonlinear activation function. Backpropagation is a supervised learning technique used by MLP during training. Weights are assigned to each neuron when a neural network is created. Backpropagation is a technique for altering the weights of neurons in order to achieve output that is closer to what is expected.MLPs are particularly well suited to projects involving tabular datasets, classification prediction challenges, and regression prediction tasks.

B. Convolution Neural Network

Data with a grid pattern, such as photographs, is processed using a convolution neural network (CNN) model. It's made to automatically learn spatial hierarchies of features. Convolution, pooling, and fully-connected layers are the three types of layers (also known as blocks) that make up a CNN.Feature extraction is performed by the convolution and pooling layers, and these extracted features are mapped into the final output by the fully connected layer. For image processing, CNN is the best option.Picture identification, image classification, object detection, and facial recognition are some of CNN's applications.

C. Recurrent Neural Networks

The output from the previous step is sent back as input to the current step in Recurrent Neural Networks (RNN). This feedback method is enabled by the RNN's hidden layer. This hidden state can be used to hold data from earlier steps in a series.RNN's'memory' aids the model in remembering all of the data that has been calculated. It then employs the same parameters for each of the inputs in order to generate the output, decreasing parameter complexity.RNNs are one of the most popular forms of neural networks, owing to their higher learning capacity and ability to execute complicated tasks like learning handwriting or language recognition. RNN is also used in a variety of other fields.

D. Our Model

We tried few traditional machine learning models like Logistic Regression, Decision Trees and Random Forest for image classification. After that, we jumped into trying a deep learning model which is Multi-layer Perceptron model which has Dense layers for image classification. Finally, we have used Convolution 2D Layers and added them before the dense layers to improve the accuracy. Our model has three Convolution 2D layers with 32,64,64 filters respectively and we have used 2 strides in each layer.

There are three dense layers after the convolution layers where the final layer is the output layer which consists of 8 neurons to classify the 8 signs. We have used “softmax” activation in the last layer and used loss function as “categorical cross entropy”, optimizer as “Adam” and metrics as “accuracy”.

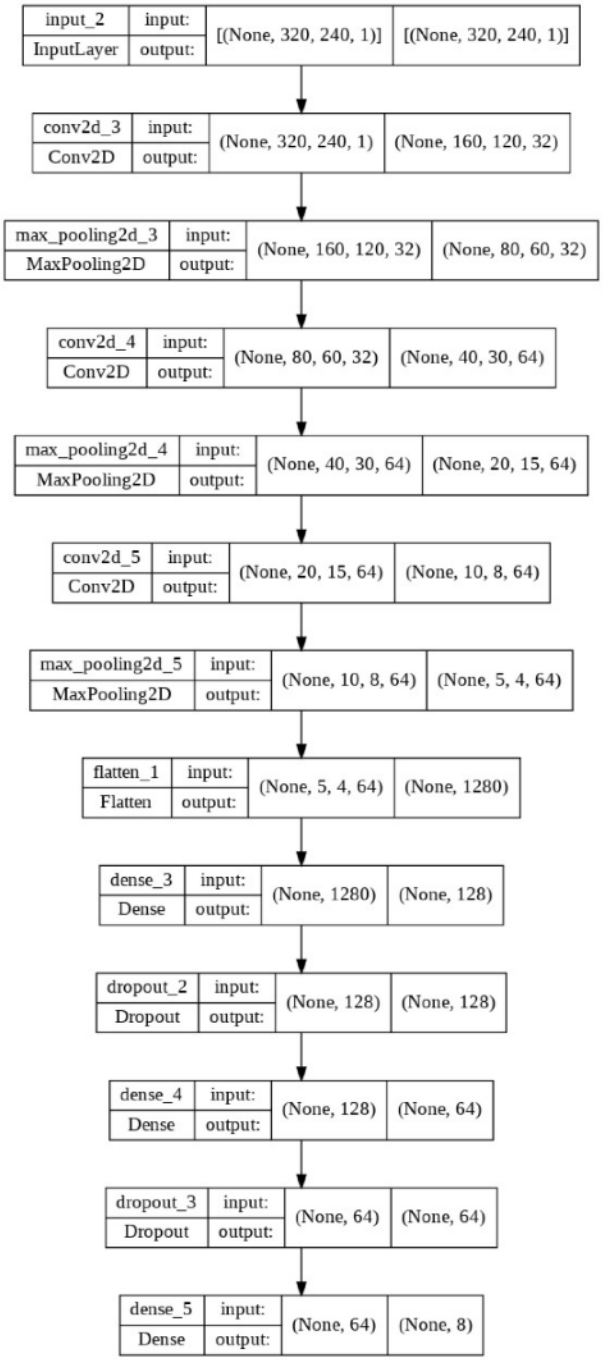

Fig. 12. Our Model

E. *tensorflow_model_optimization*

Optimize machine learning models

The TensorFlow Model Optimization Toolkit is a set of tools for improving the deployment and execution of machine learning models. Among its various applications, the toolkit supports strategies for: [19]

- Reduce cloud and edge device latency and inference costs (e.g. mobile, IoT).
- Models can be deployed to edge devices with processor, memory, power consumption, network usage, and model storage space constraints.
- Enable and optimize execution on existing hardware or new specialized accelerators.

VI. Pruning

Pruning basically refers to reduction of weights in neural network or compression procedure to the no od networks of the node in a model. [10]

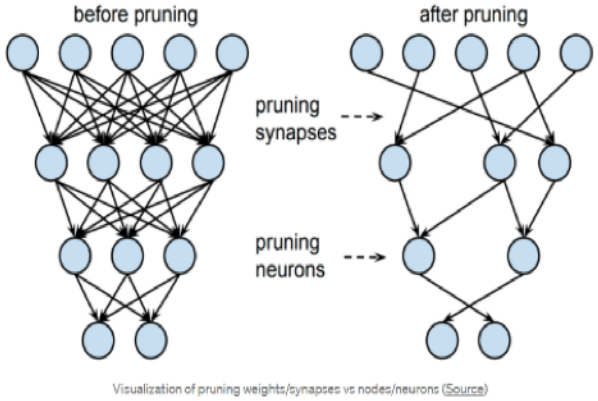

Fig. 13. Pruning

Just like how the unwanted branches are cut off from the tree while gardening the same way by pruning we eliminate unwanted weight in a model.

Deep learning’s success has been largely attributed to the development of larger and larger neural networks. This improves the performance of these models on many tasks, but it also makes them more expensive to use. Larger models require more storage space, making them more difficult to distribute. Larger models also take longer to run and may necessitate more costly gear. This is especially important if you’re putting a model into production for a real-world application.

Model compression aims to minimize the size of a model while maintaining its accuracy and performance. Pruning neural networks is a compression method in which weights from a trained model are removed.

A. What to prune?

Determining what to prune is a big difficulty in pruning. You want the parameters you remove from a model to be less useful if you're deleting weights or nodes. Different heuristics and strategies exist for evaluating which nodes are less significant and can be eliminated with little impact on accuracy. To evaluate how essential a neuron is for the model's performance, you can utilize heuristics based on its weights or activations. The idea is to eliminate as many of the less important factors as possible. The amount of the weight is one of the simplest ways to prune. A weight is effectively reset to zero when it is removed. By reducing weights that are already close to zero, or have a small magnitude, you can reduce the effect on the network. By deleting any weights below a given threshold, this can be accomplished. The L2 norm of the neuron's weights can be used to prune a neuron based on weight magnitude. In an ideal neural network, each neuron has its own set of parameters and output activations that are both relevant and not redundant. We want every neuron to be doing something unique, and we want to get rid of the ones that aren't.

B. When to prune?

One of the most important factors to consider when pruning is when it should be placed in the machine learning training/testing cycle. You should prune after training if you're utilizing the weight magnitude-based pruning method discussed in the preceding section. However, you may notice that the model's performance has degraded as a result of the trimming. Fine-tuning, or retraining the model after pruning to regain accuracy, can fix this.

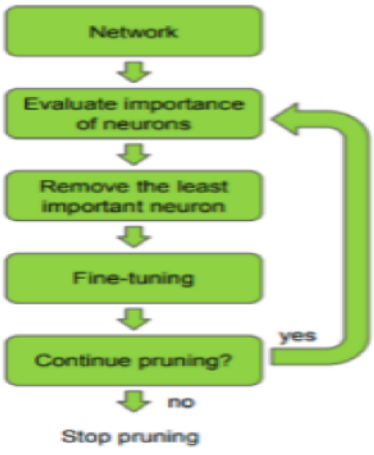

Fig. 14. When to prune

C. How to Evaluate Pruning?

When evaluating a pruning procedure, there are several factors to consider: accuracy, size, and calculation time. To determine how well the model executes its task, accuracy is required. The model's size refers to how many bytes it requires for storage. FLOPs (Floating point operations) can be used as a statistic to determine computation time. This is easier to measure than inference time, as it is independent of the model's operating system.

There is a tradeoff between model performance and efficiency while trimming. You can prune severely to create a smaller, more efficient network, but this will result in a network that is less accurate. Alternatively, you may prune lightly and end up with a network that is both highly performant and big and expensive to administer. This trade-off must be taken into account.

VII. Quantization

Quantization refers to strategies for computing and storing tensors with bitwidths that are less than floating point precision. A quantized model uses integers instead of floating point values to perform some or all of the operations on tensors. This enables the use of high-performance vectorized operations on a variety of hardware platforms, as well as a more compact model representation. In comparison to normal FP32 models, PyTorch enables INT8 quantization, which allows for a 4x reduction in model size and a 4x reduction in memory bandwidth needs. When compared to FP32 computations, hardware support for INT8 computations is typically 2 to 4 times faster. Quantization is primarily used to speed up inference, and quantized operators can only use the forward pass. [11]

K-means clustering is an unsupervised learning method that is straightforward to use. By locating clusters of pixel values, this method can be used to apply color quantization in an image. Another useful use would be the automatic classification of phonemes in a speech stream by locating formant value clusters for various speakers. In general, clustering techniques are used to group data points that are comparable. Similar data points are given a value that indicates the average of all the points in that cluster. If further data is acquired, it can be compared against the average values of other clusters to see which one is the closest. K-means is an iterative clustering algorithm that begins with a randomly assigned centroid location as the cluster's center. The closest data points to each centroid are selected in each iteration, and the overall error is obtained by adding the total distance between each point and its corresponding centroid. The centroids are then tweaked until they are representative of each cluster and the overall error is as low as possible. [12]

A. Quantization with images

Lets say you have an image of 8bpp, that has 256 different levels. It is a grayscale image and the image looks something like this. [21]

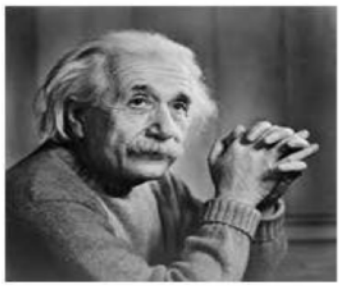

Fig. 15. 256 Gray Levels
[21]

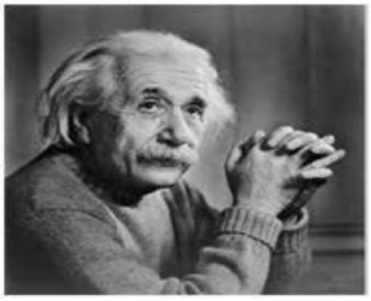

Fig. 16. 64 Gray Levels
[21]

Fig. 17. 2 Gray Levels
[21]

B. When to Quantize?

Quantization can be classified into two types: post-training quantization and quantization aware training. Although quantization conscious training is frequently superior for model accuracy, start with post-training quantization because it's easier to utilize.

Post-training quantization:-
Post-training quantization includes general techniques to reduce CPU and hardware accelerator latency, processing, power, and model size with little degradation in model accuracy. These techniques can be performed on an already-trained float TensorFlow model and applied during TensorFlow Lite conversion.

Quantizing weights:
Weights can be converted to types with reduced precision, such as 16 bit floats or 8 bit integers. We generally recommend 16-bit floats for GPU acceleration and 8-bit integer for CPU execution.

Improve latency, processing, and power usage, and get access to integer-only hardware accelerators by making sure both weights and activations are quantized.

In our Project we have utilized quantization to mainly convert the float values to integer below is the piece of code which we used in our project to do so.

```
converter = tf.lite.TFLiteConverter
.from_keras_model(model_for_export)
converter.inference_input_type = tf.int8
# or tf.uint8
converter.inference_output_type = tf.int8
# or tf.uint8
```

VIII. Dense Layers

A dense layer in a neural network is one that is deeply connected to the layer before it, meaning that the layer's neurons are connected to every neuron in the layer before it. In artificial neural network networks, this layer is the most widely utilized layer.In a model, the dense layer's neuron receives input from every neuron in the preceding layer, and the dense layer's neurons conduct matrix-vector multiplication. The row vector of the output from the preceding layers is identical to the column vector of the dense layer in matrix vector multiplication. In matrix-vector multiplication, the row vector must have the same number of columns as the column vector. [16]

$$\mathbf{Ax} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}.$$

Fig. 18. Dense Layer

IX. Activation Function

Basically, the SELU activation function multiplies scale (¿ 1) with the output of the tf.keras.activations.elu function to

ensure a slope larger than one for positive inputs. Activations that are more complex than a simple TensorFlow function (eg. learnable activations, which maintain a state) are available as Advanced Activation layers, and can be found in the module tf.keras.layers.advanced_activations. These include PReLU and LeakyReLU. If you need a custom activation that requires a state, you should implement it as a custom layer. [13]

A. Relu

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning. ReLU is half rectified (from bottom). [22]

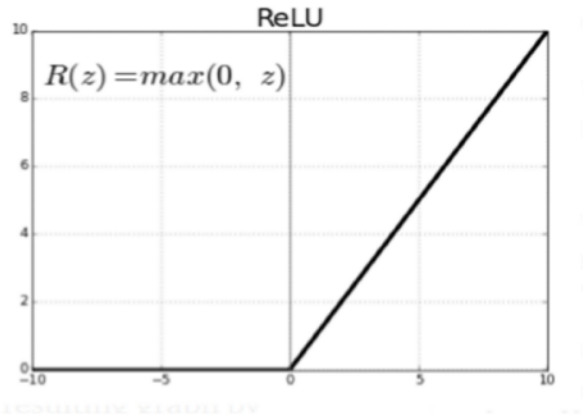

Fig. 19. Relu
[22]

f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero. But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately. [22]

The advantages of using ReLU as an activation function are as follows: [23]

- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions.
- ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.

B. Softmax :

Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class. [24]

X. Max Pooling

Pooling that selects the maximum element from the region of the feature map covered by the filter is known as max pooling. As a result, the output of the max-pooling layer would be a feature map with the most prominent features from the previous feature map. [14]

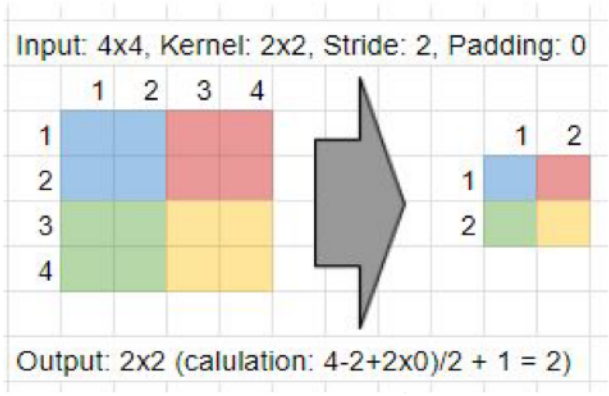

Fig. 20. Max pooling
[17]

XI. Convolution Layers

In convolutional neural networks, the major building elements are convolutional layers. A convolution is the basic process of applying a filter to an input to produce an activation. When the same filter is applied to an input multiple times, a feature map is created, displaying the positions and strength of a recognized feature in an input, such as an image. The capacity of convolutional neural networks to learn a large number of filters in parallel particular to a training dataset under the restrictions of a certain predictive modeling problem, such as image classification, is its unique feature. As a result, extremely precise traits appear on input photographs that can be identified everywhere. [15]

XII. TF lite

TensorFlow Lite is an open-source, product ready, cross-platform deep learning framework that converts a pre-trained model in TensorFlow to a special format that can be optimized for speed or storage.

The special format model can be deployed on devices like mobiles using Android or iOS or Linux based embedded devices like Raspberry Pi or Microcontrollers to make the inference.

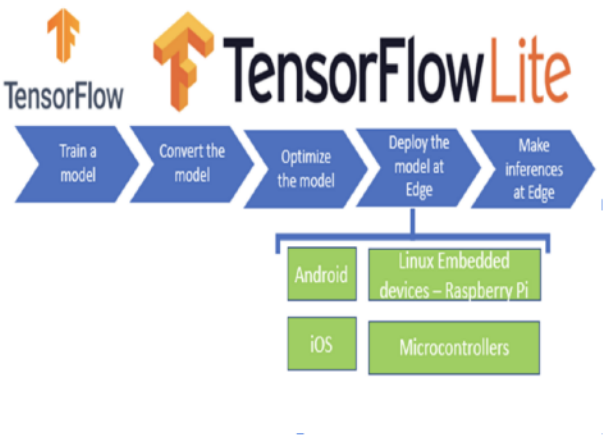

Fig. 21. TF lite

TF lite model is a special format model efficient in terms of accuracy and also is a light-weight version that will occupy less space, these features make TF Lite models the right fit to work on microcontrollers as they have a very less flash memory size.Tensorflow Lite Converter converts a Tensorflow model to Tensorflow Lite flat buffer file(.tflite).**?**

The below image shows the tensor flow lite conversion process

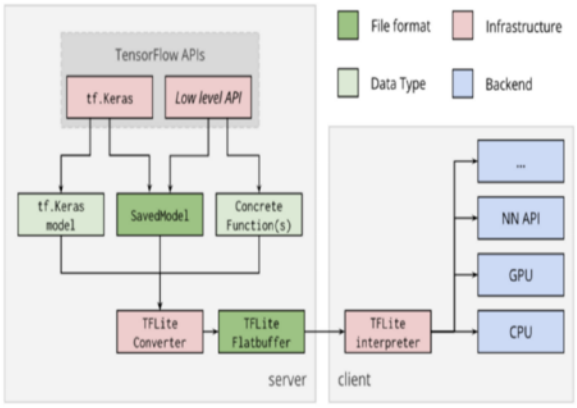

Fig. 22. TF lite

XIII. Results

We could see that, the training accuracy keeps on increasing while reaching the last epoch and similarly for the case of validation accuracy, the accuracy is keeps on increasing till the last epoch, which indicates that the model is doing good.

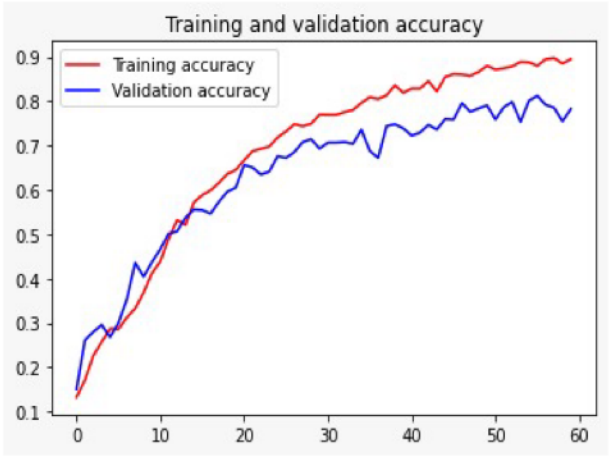

Fig. 23. Training and Validation Accuracy

The plot of training loss and validation loss. We could see that, the training loss keeps on decreasing while reaching the last epoch and similarly for the case of validation loss, the loss keeps on decreasing till the last epoch (but irregularly), which indicates that the model is doing good.

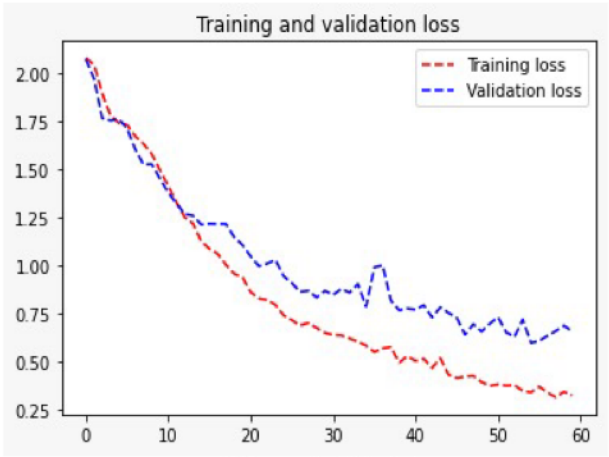

Fig. 24. Training and Validation Loss

Accuracy and Loss:

After training for 60 epochs with batch size of 100, we evaluated the model to check for accuracy and the loss. We got the accuracy as 78.87% and the loss is 0.61. The below graph shows the varying accuracy and loss both while training and the validation.

Pruned model Accuracy and loss: After pruning the deep leaning model and training with the given data, we could get the following graphs.

A. *Pruning*

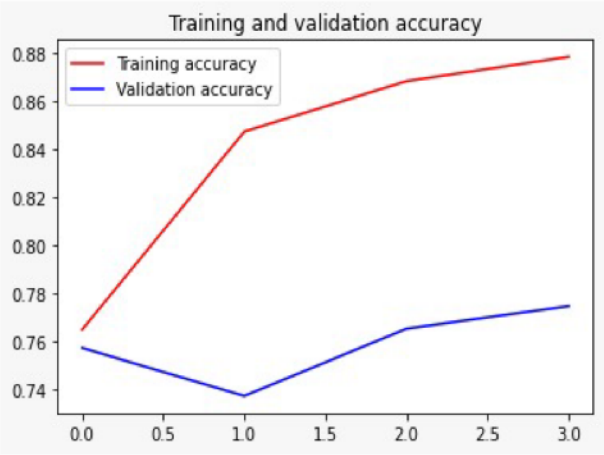

Fig. 25. Pruning accuracy

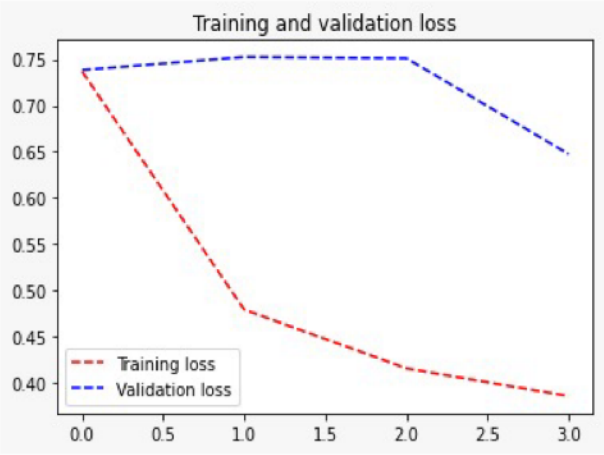

Fig. 26. Pruning loss

XIV. DEPLOYMENT

We converted our Tensor flow model to TFlite file. We then transferred the TFlite file into the Portenta along with the labels file containing the label names of all the 8 road signs used. Using Open MV software we loaded the TFlite and label files transferred into the Portenta for real time inference by using the below code.

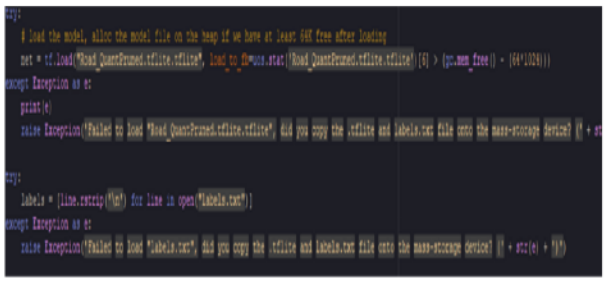

Fig. 27. Deployment

Once the files have been loaded we then used the net.classify function to get the confidence values of all the 8 classes. After getting the confidence values we selected the class with max confidence value and displayed it on the serial monitor using the below code.

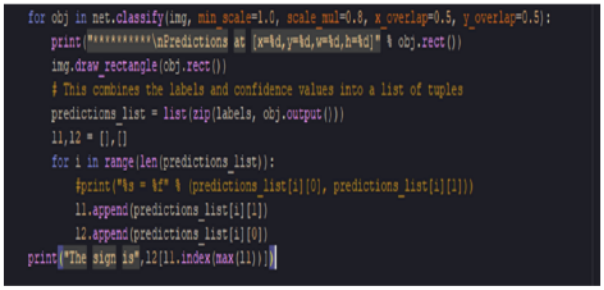

Fig. 28. Deployment

XV. FUTURE IMPROVEMENT AND SCOPE

Increase the training data size and also improve the model by trying out various other combinations of dense and convolution layers. The quality of training data can be further improved by taking the images with different backgrounds, varying brightness and varying angles. Another major improvement would be to add an unidentified class and train the class with images containing no signs. This mitigates the issue where the Portenta predicts the image as one of the road sign even when there is no sign in the image used for prediction.

This can be further developed and used in expert systems, such as automatic driving systems as traffic sign detection plays a major role in recognising the traffic signs and making decisions based on it as it can lead to occurrence of traffic accidents with the rise in demand for the intelligent vehicles. It can also be incorporated into maps to get all the road signs information in a particular area and use that information while making decisions and informing the user while traveling.

REFERENCES

[1] https://uk.rs-online.com/web/generalDisplay.html?id=ideas-and-advice/microcontrollers-guide

[2] http://store.arduino.cc/products/arduino-nano-33-ble
[3] https://docs.arduino.cc/hardware/portenta-h7
[4] http://store.arduino.cc/products/portenta-h7
[5] https://store-usa.arduino.cc/products/arduino-portenta-vision-shield-ethernet
[6] https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/
[7] https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model
[8] https://www.ibm.com/docs/en/spss-modeler/18.0.0?topic=networks-neural-model
[9] https://analyticsindiamag.com/top-5-neural-network-models-for-deep-learning-their-applications/
[10] https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9
[11] https://pytorch.org/docs/stable/quantization.html
[12] https://www.projectrhea.org/rhea/index.php/Quantization_and_Classification_using_K-Means_Clustering
[13] https://keras.io/api/layers/activations/: :text=Basically
[14] https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer
[15] https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
[16] https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/
[17] https://datascience.stackexchange.com/questions/67334/whats-the-purpose-of-padding-with-maxpooling
[18] https://store-usa.arduino.cc/products/portenta-h7?selectedStore=us
[19] https://www.tensorflow.org/model_optimization
[20] https://github.com/ebgoldstein/MokaPotNet/blob/main/src/MokaPotNet_portenta_sparse.ipynb
[21] https://www.tutorialspoint.com/dip/concept_of_quantization.html
[22] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
[23] https://www.v7labs.com/blog/neural-networks-activation-functions
[24] https://machinelearningmastery.com/softmax-activation-function-with-python/
[25] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.